PENNANT

Summary Version

This is version 1 of this summary.

The benchmark tests were run with the tagged version 0.9 of PENNANT on https://github.com/lanl/PENNANT . (Note that, between that version and the December 2017 head, the source code is unchanged; only tests and documentation have been added.)

Purpose of Benchmark

Test performance of hydrodynamics on general unstructured meshes in 2D (arbitrary polygons).

Characteristics of Benchmark

PENNANT is a hydrodynamics mini-app which operates on 2-D unstructured finite element meshes containing arbitrary polygons. It makes heavy use of indirect addressing and irregular memory access patterns. It is written in standard C++ and supports MPI and OpenMP parallelism.

More detailed documentation can be found in the doc directory of the PENNANT distribution.

Building the Benchmark

A simple Makefile is provided in the top-level directory for building the code. Before using it, you may wish to edit the definitions of CXX and CXXFLAGS to specify your desired C++ compiler and flags, and to choose between optimized/debug and serial/OpenMP/MPI builds. Then a simple "make" command will create a build subdirectory and build the PENNANT binary in that directory.

PENNANT has been tested under GCC, PGI, Intel, and IBM compilers. Building under other compilers should require at most minor changes.

Running the Benchmark

Several test problems are provided in subdirectories under the top-level test directory. Each subdirectory contains a small text file <testname>.pnt containing run parameters for that test case. The command line

```
pennant <testname>.pnt
```

is used to run a test in serial mode. If running under MPI, this should be preceded by mpirun or similar command as appropriate on your system. Use the -np (or equivalent) argument to change the number of MPI ranks, and the code will determine a suitable decomposition. Similarly, for OpenMP, specify the OMP_NUM_THREADS environment variable.

The problem used for baseline runs and FOM calculations on Vulcan is sedovflatx40 (“1/4 of Sequoia”). The corresponding CORAL-2 target problem is sedovflatx120. The target problem is larger than their respective baseline by a factor of 9 (note that the xN suffix describes resolution in each dimension, so total problem size grows as N^2). The test directory also includes smaller, single-node versions of sedovflat, as well as several other tests that may be useful for debugging.

The outputs from the baseline runs on Vulcan can be found in the test/sample_outputs/vulcan directory in the PENNANT distribution.

Figure of Merit (FOM)

The FOM for PENNANT benchmarks is total zones processed per second, which can be computed from three lines in PENNANT standard output:

- number of zones in mesh: in the line ‘Zones:’ near the top
- number of cycles run: in line ‘cycle =’ near the bottom
- run time: in line ‘hydro cycle run time=’ near the bottom (note that this time excludes initialization and finalization stages)

For the benchmark problems the FOM is:

sedovflatx40:

FOM = zones * cycles / time
= 414720000 * 125001 / 5.216800e+02
= 9.9404e+10

Verification of Results

Results can be verified by comparing the final "energy check" diagnostics, printed near the end of the standard output, to those in the sample output file. All three quantities listed (total, internal, and kinetic energy) should match to within a relative error of 1.e-5.

Additional Note

The purpose of the PENNANT benchmark is to demonstrate performance on an application using general unstructured meshes, that is, meshes which contain arbitrary polygons with arbitrary connectivity between them. Any code changes made by the vendor must preserve this capability. That is, although the leblanc* and sedovflat* benchmark problems happen to use meshes with an underlying Cartesian structure, the code may not be modified in any way that takes advantage of this structure. (This can be verified by ensuring that any modified code can correctly run the nohpoly test problem, in which the mesh is truly unstructured.)

Vendors are free to make other modifications to the mesh (different data structures, element numbering, MPI decomposition, ...) so long as support for general unstructured meshes is preserved.